C. Remarks

Rejections under 35 U.S.C. §103:

Claims 1 – 18 are rejected under 35 U.S.C. §103(a) as obvious in view of the single reference Grimm et al. (US Patent 6,317,868).

Claim 1:

The present invention, as set forth in Claim 1, requires an "execution request" be sent to the claimed "security server system" as a precondition for enabling the program that is the object of the execution request to even begin execution. This "execution request" is defined in the claim as including "an identification of a program load request, request context related data, and a secure program signature."

Claim 1 also requires the "security server system" contain "a database storing sets of pre-qualified program signatures" against which to validate the "secure program signature" presented in an "execution request."

Finally, Claim 1 also requires "request context related data" to be sent to the "security server system" as part of the execution request.

Grimm Does Not Teach or Suggest Use of the Claimed "Secure Program Signature":

The Action, in support of the obviousness rejection, equates the claimed "secure program signature" with a "security identifier" described in Grimm. Without this equivalence, as properly acknowledged by the Action, Grimm does not teach or suggest the present invention.

Rather than being equivalent, the "security identifier" of Grimm and the "secure program signature" of Claim 1 are, when fully considered, quite different technical entities used in completely different manners to achieve distinctly different results. In brief, the "security identifier" of Grimm is merely a variable with an externally assigned value established to either identify a type of software component or identify a security operation the Grimm is required to perform at some point in the execution of that component. The

Grimm "security identifier" cannot uniquely identify a software component or even be used prior to execution of the software component.

The "secure program signature" of Claim 1, in contrast, is a specific, unchangeable value unique to a particular program; it is a secure digital signature computed from the image of the program itself. The "security server system" of Claim 1 must, on its own, determine whether a uniquely identified program will be permitted to even begin execution based on the secure signature of the program image and data describing the operating context within which the program load request was issued (the claimed "request context related data").

The Grimm system describes preprocessing of software components to functionally inject code into a software component that, when subsequently executed, forces "the modified software component ... to adhere to the security policy at the site" where the component is executed (Col. 4, ll 18 - 22). The entire concept of Grimm is to allow certain identifiable security concerns to be addressed on a site-specific basis. No concept of stopping execution; just curtailing ongoing execution to align behavior with local security concerns. Consequently, it is the injected code, not the original software component, that is coded to contain whatever site-specific security identifier is deemed applicable by the preprocessor. The injected code, during execution of the modified component, forwards the assigned security identifier to an external security service for evaluation and modifies the behavior of the component dependent on the result.

Grimm teaches that an instance of injected code apparently contains both a security identifier that is "associated with the software component itself" (Col. 5, ll 42 - 44) and a security identifier that will instruct the security service as to "when and how to perform access checks, protection domain transfers, and auditing during execution of the modified software component" (Col. 5, ll 44 - 47).

In determining the value of these security identifiers, the Grimm pre-execution parser only evaluates the component for operations that it determines will "require access control checks, protection domain transfer and auditing" (Col. 5, ll 15 - 20). The parser then injects executable code pre-loaded with the security identifiers that will specify the security checks to be performed by some security policy service when the software component is subsequently executed. Grimm explicitly describes the security identifiers as internally defined values:

a) that identify the security operation to be performed when the injected code is executed (Col. 6, ll 28 - 31),

b) as a reference to another object (Col. 6, ll 31 - 33),

c) to some default object security identifier (Col. 6, ll 33 - 37), or

d) as "associated with an object's name" in some identified namespace (Col. 6, ll 37 - 41).

In the case of (a), any number of different components may be injected with the same security identifier in order to have the same security operation performed. In the case of (b), both the injected component and the referenced object evidently use the same security identifier to request the same security operation. In the case of (c), all defaulted components use the same security identifier. In the case of (d), Grimm makes clear that the same security identifier may be "associated" with the names of multiple, otherwise unrelated components.

As these are the only described basis for selecting security identifiers, and all demonstrate relative values, the only reasonable conclusion is that the security identifier "associated" with the software component is at most associated by name only. Plainly, then, Grimm contemplates different actual software components having the same "name" for purposes of identifying software components to the security service.

As for the selection of the second identifier, the specific value of the injected security identifier reflects a <u>site-dependent</u> policy evaluation of the security requirements implicated by the operation of the component at the location where the code is injected. Thus, the value of the security identifier is not a unique identifier of the software module into which it is injected, but rather a policy determined identifier of a generic security evaluation that must be made at this particular point in the execution of the software component.

Grimm thus inherently establishes that the security identifiers issued to the security service serve only to generically identify a software component and identify a particular external security evaluation to be performed. An assured unique identification of a software component is never made by Grimm and certainly is never provided to the security service.

Furthermore, injected security identifiers are and must be changeable values as required to support protection domain transfers. That is, according to Grimm, an injected security identifier is replaced with another domain specific security identifier when the component functionally moves by execution to a different protection domain (Col. 7, ll 7 - 10). Here, Grimm expressly teaches that the "subject" of a security identifier, *i.e.*, a parameter object which may be of or in another software component, can have its security identifier changed on transition between domains. That is, Grimm teaches that the security identifier "associated" with a software component can and will be changed in the execution of the software component. Thus, inherently, Grimm cannot support protection domain transfers if both the security identifier "associated" with a software component and the security identifier that identifies the requested security operation are not dynamically modifiable.

Conversely, the present invention, as set forth in Claim 1, literally requires identification of a program by a "secure program signature." Even without more, a person of ordinary skill in the art will recognize the partial term "program signature" as defining a digital signature value uniquely identifying a particular program. The full term "secure program signature" is readily understood as requiring a digital signature generated from the program image using an encryption type algorithm to ensure uniqueness.

Furthermore, this understanding of the meaning of "secure program signature" is fully consistent with the functional definition of that term as provided in the present specification. Consistently, in at least paragraphs 46, 52, 53, and 61 of the present specification, the "secure program signature" is described as a "secure hash calculated signature of the program image" and the result of "generation of a secure signature, preferably SHA-1 based, for any executable image loaded from either a local or remote filesystem."

The "secure program signature" is therefore properly a singular fixed value that is at least statistically unique to the program for which it was generated.

Consequently, the "secure identifier" of Grimm is not a teaching of the "secure program signature" required by Claim 1.

The described use of the "secure identifier" by Grimm is also not a suggestion of the claimed "secure program signature." Grimm explicitly requires the software components of an application to be executed in order to issue security identifiers. Even then, the evaluation of the security identifiers results only in qualifying access to other application internal objects or objects that the application, as then executing, may reference; these objects being identified by execution of the injected code as arguments coupled with a security identifier sent for evaluation (Col. 6, 20 - 28). Even then, Grimm requires these security identifiers to be modifiable as required to implement protection domain transfers.

Plainly, Grimm itself does not suggest the wholesale reconstruction that would be necessary to send security identifiers <u>prior</u> to execution, to identify the referenced argument objects without executing the software component, and to implement a decision of whether the software component can be executed in the first instance. Furthermore, Grimm does not express any motivation to even consider making such changes. Inherently, Grimm accepts that the software components, with injected code, will execute and that the security identifiers are alterable dynamically as needed. Determining not to execute a component before it is ever loaded or not supporting the protection domain transfer feature is simply never contemplated in Grimm.

For at least this reason, Grimm fails to teach or suggest the present invention as set forth in Claim 1.

<u>Grimm Does Not Teach or Suggest the Claimed "Pre-qualified Program Signatures" Database:</u>

Given that Grimm is acknowledged in the Action as <u>not</u> teaching a database of security identifiers, the Action relies on the inductive reasoning that a person of ordinary skill would understand that a database of security identifiers, or reasonable equivalent, must be present in the Grimm security service to support the security service functions described in the reference. Based on the assertion that the claimed "secure program signatures" are just "security identifiers," the conclusion is reached that the presumed database stores claimed "secure program signatures."

The assertion that the Grimm "security service" must contain some mechanism for recognizing provided "security identifiers" and that a person of ordinary skill in the art might reasonably consider using a database for that purpose is likely correct. Any assertion that

Grimm teaches or suggests that the database must equally store"pre-qualified program signatures," as required by Claim 1, is not.

Specifically, without any reason to even consider using fixed, unique values to identify program images, as established above, Grimm provides no further teaching or suggestion that the "security server system" resident database contain "pre-qualified program signatures." Again, as established above, the partial term "program signature" is properly recognized as referring to a unique value identifying a specific program. Qualifying the term as "pre-qualified" recognizes that the database is established and defined in advance specifically in regard to a set of "secure program signatures." This interpretation is entirely consistent with the description of the database and its contents as presented in the present specification:

> A signature database 72 locally provided on the security appliance 60 is also accessible to the control program 64. Preferably, the signature database stores secure, SHA-1 based signatures for an administratively determined set of executable programs, including associated executable library files. (¶ 53.)

While Grimm might suggest the use of a database for identifiers, Grimm does not teach or suggest a "security server system" resident database that specifically stores "pre-qualified program signatures."

For this reason as well, Grimm fails to teach or suggest the present invention as set forth in Claim 1.

<u>Grimm Does Not Teach or Suggest Sending "Request Context Related Data":</u>

The present invention, as set forth in Claim 1, requires the evaluation of the secure program signature <u>prior</u> to the execution of the program in combination with "request context related data." This data is literally a set of data related to the context of the request that transmits the secure program identifier. In the field of computer science, the term "context" has a well-understood meaning: the circumstances under which a device is being used or the circumstances or settings which determine, specify, or clarify the meaning of an event. Even in general use, "context" is defined consistently: the interrelated conditions in which something exists or occurs (Merriam-Webster Online Dictionary).

The present specification describes the "request context related data" as the authentication and authorization information contained within the process that originates the program load request (¶ 42).

At most, Grimm teaches providing data related to objects within the execution context of the software component. This context does not exist prior to execution of the software component. Consequently, the Grimm reference does not teach or suggest the sending of a fixed, unique "secure program signature" in combination with the launch requesting context <u>before</u> the program executes.

For the forgoing reasons, the Grimm reference fails to teach or suggest the present invention. Reconsideration of the rejection of Claim 1 is respectfully requested.

<u>Claims 2:</u>

This claim adds the further limitation that a "module" executes on the host computer to recognize a program load request and responsively generate an "execution request" which is then sent to the security server system. This module is similar to the preprocessor of Grimm to the extent that it effectively intercepts load requests. Unlike the Grimm preprocessor, the claimed "module" is the source of the "execution requests," including secure program signatures, that are sent to the security server system for evaluation. Further unlike Grimm, the claimed "module" is "operative to permit or deny said program load request." That is, the claimed module is capable of precluding the load requested program from ever beginning execution.

Nothing in Grimm teaches or suggests that any security request be sent to its security service <u>prior</u> to the execution of a software component. Grimm does teach that the continued execution of a software component maybe affected by the results of a security check (Col. 6, 51 - 57). As described, a Grimm software component will execute up and through an injected code section to initiate a security check. The result of a security check is returned not to the operating system or the preprocessor module, but to the software component itself as appropriate to handle the potential security exception condition.

Consequently, in addition to not sending a security request <u>prior</u> to the execution of a software component, Grimm also does not teach or suggest that such an preemptive security check be used to prevent a software component from even beginning execution.

Claim 2 is therefore not obvious in view of Grimm. Reconsideration of the rejection of Claim 2 is therefore respectfully requested.

Claims 3 – 8:

Claims 3 - 8 are dependent on Claim 2. Applicants respectfully assert that these claims are not obvious for at least the reasons presented above. Reconsideration of the rejection of Claims 3 - 8 is therefore respectfully requested.

Claim 9:

Independent Claim 9 calls for "a module installed as a component of a host computer system" and a "security server." Notably, the Action equates the claimed "module" with a Grimm software component. The claimed module, however, is operative to:

> intercept system calls to load an execute program for execution

and

> generate a security request containing a predetermined load request, associated authentication data and access attributes and a target secure program signature of an executable program identified by said predetermined load request

Grimm, in contrast, uses a pre-processor module to intercept the loading of software component. The modified software component, however, is the entity that, when subsequently executed, sends out a security identifier. The fundamental difference, as discussed above, is that the claimed invention issues a security request prior to the load requested program being executed.

Furthermore, this preemptive security request is required by the claim to send the "secure program signature" and "associated authentication data and access attributes" in the load request. Grimm does not teach or suggest sending, preemptively or otherwise, a "secure program signature." Grimm also does not teach or suggest ever sending or considering the "authentication data and access attributes" associated with the load request itself. These are security considerations not even contemplated by Grimm; that is, Grimm assumes that the software component will be loaded and will be executed.

Claim 9 also requires the "security server" to consider the "secure program signature" against a database of "pre-qualified secure program signatures" and to evaluate the security request and "authentication data and access attributes" associated with the load request.

As established above, Grimm does not teach or suggest sending a "secure program signature," let alone a server resident database of "pre-qualified secure program signatures." Likewise, Grimm does not teach or suggest even considering load request "authentication data and access attributes."

Accordingly, Applicants respectfully assert that Claim 9 is not obvious in view of Grimm. Reconsideration of the rejection of Claim 9 is respectfully requested.

<u>Claims 10 - 13:</u>

Applicants respectfully assert that dependent Claims 10 - 13 are not obvious in view of Grimm for at least the same reasons identified above in regard to independent Claim 9. Reconsideration of the rejection of Claims 10 - 13 is respectfully requested.

<u>Claims 14:</u>

Independent method Claim 14 defines a method operative at the point that a load request is intercepted on a host computer. The claimed method requires:

> b) determining authorization data and access attributes associated with said load request;
> c) generating a secure signature for said program;

and

> d) providing ... an identification of said load request, said authorization data and access attributes and said secure signature, to a security server

These steps occur before the program identified by the load request is executed. That is, based on the evaluation of the security request containing the information in step (d), the security server returns a "security request response" that determined whether the program will be executed or not. Specifically, Claim 14 requires:

> e) selectively enabling performance of said load request dependent on said security request response.

Grimm, in clear contrast, requires a software component to be loaded and to have already begun execution before a security identifier is ever sent to the Grim security service. Furthermore, nothing in Grimm teaches or suggests the steps of generating the "secure program signature," determining "authorization data and access attributes," or sending both to the security server system for policy evaluation.

The substantive differences in the steps required by Claim 14 and the point at which they are performed is not taught or suggested by the Grimm reference. Claim 14 is therefore not obvious in view of the Grimm reference. Reconsideration of the rejection of Claim 14 is respectfully requested.

Claims 15 - 18:

Applicants respectfully assert that dependent Claims 15 - 18 are not obvious in view of Grimm for at least the same reasons identified above in regard to independent Claim 14. Reconsideration of the rejection of Claims 15 - 18 is respectfully requested.

Conclusion:

In view of the above Amendments and Remarks, Applicants respectfully assert that Claims 1 through 18 are properly in condition for allowance. The Examiner is respectfully requested to take action consistent therewith and pass this application on to issuance. The Examiner is respectfully requested to contact the Applicants' Attorney, at the telephone number provided below, in regard to any matter that the Examiner may identify that might be resolved through a teleconference with the Examiner.

Respectfully submitted,

Date: 3/10/2006

By: [illegible]
Gerald B. Rosenberg
Reg. No. 30,320

NEWTECHLAW
285 Hamilton Avenue, Suite 520
Palo Alto, California 94301
Telephone: 650.325.2100